Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

PROTOTYP MOBILAPP

för Crosskey Banking Solutions

Maiken Söderlund

2020:35

Datum för godkännande: 02.06.2020
Handledare: Agneta Eriksson-Granskog

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Maiken Söderlund
Arbetets namn:	Prototyp mobilapp för Crosskey Banking Solutions
Handledare:	Agneta Eriksson-Granskog
Uppdragsgivare:	Crosskey Banking Solutions

Abstrakt:

Syftet med uppdraget var att utveckla en demomobilapplikation som kan demonstrera funktionalitet och design för potentiella kunder. Uppdraget var beställt av Crosskey Banking Solutions.

Kraven för vad som skulle ingå togs fram med hjälp av möten och framställning av en kravspecifikation. Applikationen utvecklades med kombinationen Angular och Ionic.

Resultatet blev en första version av en demoapplikation, som enkelt kan vidareutvecklas.

Nyckelord (sökord):

prototyp, mobilapp, demo, angular, ionic

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2020:35	1458-1531	Svenska	24

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
31.05.2020	13.05.2020	02.06.2020

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Information systems
Author:	Maiken Söderlund
Title:	Prototype Mobile App for Crosskey Banking Solutions
Academic Supervisor:	Agneta Eriksson-Granskog
Technical Supervisor:	Crosskey Banking Solutions

Abstract:

The purpose of the assignment was to develop a demo mobile application that can demonstrate functionality and design for potential customers. The assignment was requested by Crosskey Banking Solutions.

The requirements for what was to be included were drawn up by means of meetings and preparation of a requirements specification. The application was developed with the combination of Angular and Ionic.

The result was a first version of a demo application, which can easily be further developed.

Key words:

prototype, mobile app, demo, angular, ionic

Serial number:	ISSN:	Language:	Number of pages:
2020:35	1458-1531	Swedish	24

Handed in:	Date of presentation:	Approved on:
31.05.2020	13.05.2020	02.06.2020

INNEHÅLLSFÖRTECKNING

1. INLEDNING 5
1.1 Syfte 5
1.2 Metod 5
1.3 Avgränsningar 6

2. DEFINITIONER 7
2.1 Angular 7
2.2 Data Mocking 8
2.3 Hybridapplikation 8
2.4 Ionic 8
2.5 Jenkins 8

3. KRAV 9
3.1 Vad ingår i projektet? 9

4. EN FÖRSTA GRUND 10
4.1 Bakgrund 10
4.2 Utveckling av applikationen 10
4.3 Skapande av Jenkins bygge 13

5. HANTERING AV MOCKDATA 14
5.1 Val av metod 14
5.2 Databas i minnet 15

6. DESIGN OCH LAYOUT 18
6.1 Teman 18
6.2 Navigering 18
6.3 Val av språk 20
6.4 Resultat 20

7. SLUTSATSER 22
7.1 Framtida utveckling 22
7.1.1 Open banking integrering 22
7.2 Reflektioner 23

KÄLLOR 24

1. INLEDNING

1.1 Syfte

Syftet med uppdraget är att utveckla en mobilapplikation med ändamålet att demonstrera funktionalitet och design som kan erbjudas till potentiella kunder. Min uppdragsgivare är Crosskey Banking Solutions, som har applikationer som Ålandsbanken Finland och Ålandsbanken Sverige på marknaden. Mobilapplikationen ska kunna användas utan internetuppkoppling, vilket innebär att alla data redan ska finnas tillgänglig i applikationen.

1.2 Metod

Mitt projekt började med ett möte med min handledare och min produktbeställare från Crosskey. Tillsammans gjorde vi upp om vad som skulle ingå i projektet. Efter det sammanställde jag informationen genom att ta fram en första kravspecifikation. Min handledare hjälpte mig att skapa upp projektet, sedan har jag utvecklat självgående. Efter att en viss tid av arbete har gjorts så demonstrerade jag applikationens funktionalitet och diskuterade vad som skulle fokuseras på därnäst med handledaren och ibland även beställaren.

Jag har haft möjligheten att jobba för Crosskey som trainee ett tag innan detta projekt, så jag har redan kunskap om hur vi i teamet arbetar och hur koden är standardiserad. Detta med hjälp av min kunskap från mina akademiska studier ser till att jag kan lösa uppdraget.

Applikationen byggs upp med hjälp av webbramverket Angular och ramverket för applikationsutveckling Ionic. Min utvecklingsmiljö är IntelliJ, eftersom jag inom Crosskey är van vid det verktyget.

1.3 Avgränsningar

I projektet ingår inte att producera en komplett applikation, utan endast en stadig dynamisk grund som enkelt kan utökas med funktionalitet. Framställning av mockdata och styling av applikationen ingår, men kommer inte vara fullständig. Målet är att kunna uppvisa vad som är möjligt i en modern hybridapplikation inom bankvärlden. En senare idé var att applikationen även ska ha en början till en koppling mot Crosskeys Open Banking-lösning, vilket jag vid denna tidpunkt inte har färdigställt ännu.

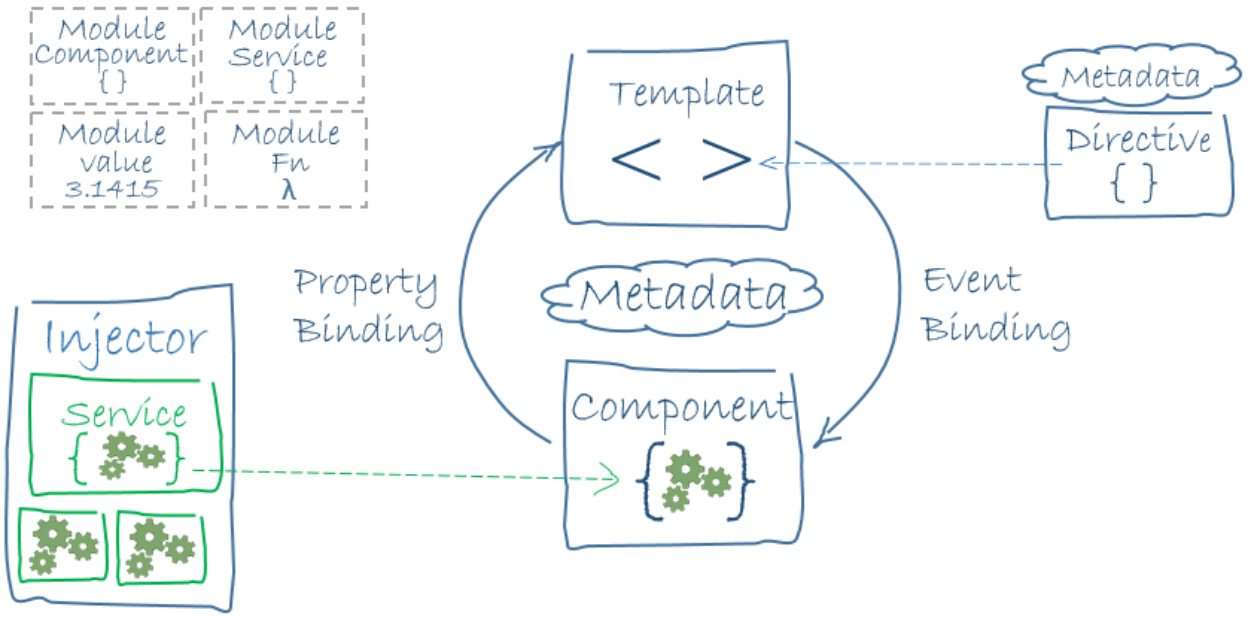
2. DEFINITIONER

2.1 Angular

Angular är ett webbramverk som är TypeScript-baserat (Angular, 2019). TypeScript är ett programmeringsspråk som är utvecklat av Microsoft. Föregångaren till Angular är AngularJS, som är ett webbramverk baserat på JavaScript. Angular är en fullständig omskrivning av AngularJS, med skillnader som:

- Användning av en hierarki av komponenter istället för en kontext (“scope”)
- En annorlunda uttryckssyntax
- Modularitet, dvs. en stor användning av moduler

Figur 1 förklarar arkitekturen av en Angular-applikation. De huvudsakliga byggklossarna är moduler, komponenter, mallar, metadata, databindning, direktiv, tjänster och beroende injektion (Angular, 2019).



Figur 1. Arkitekturen av en Angular-applikation (Angular, 2019).

2.2 Data Mocking

"Mockdata" är falska data, som ofta används i syfte att testa funktionalitet i olika scenarier. Dessa data införs artificiellt till mjukvaran. Detta tillvägagångssätt kallas för "mocking". En stor fördel med mockdata är möjligheten att simulera fel och omständigheter som är svåra att skapa i en riktig miljö (Hamilton, 2016). I projektet så använder jag mockdata för att visa upp applikationen i en falsk kunds perspektiv.

2.3 Hybridapplikation

En hybridapplikation är en blandning mellan en native lösning och en webblösning. Detta innebär att applikationens kärna är skriven med hjälp av webbtekniker (HTML, CSS och JavaScript), som sedan inkapslas i en native-applikation. Genom användning av plugin har applikationen full tillgång till den mobila enhetens funktioner (Griffith, 2019).

2.4 Ionic

Ionicramverket är en uppsättning utvecklingsverktyg för hybrid mobilapplikationsutveckling, skapad 2013. Den senaste versionen av Ionic gör det möjligt för utvecklare att välja vilket användargränssnittsramverk de vill använda tillsammans med Ionic, exempelvis Angular, React eller Vue.js. Ionic erbjuder verktyg och tjänster, som med moderna webbutvecklingstekniker hjälper utvecklare att skapa hybrida mobila, skrivbords- och progressiva webbapplikationer (Ionic, 2019).

2.5 Jenkins

Jenkins är en automatiseringsserver skriven i Java och är öppen källkod. Det är ett verktyg som stödjer automation för många olika slags utvecklingsuppgifter. Jenkins tillhandahåller ett snabbt och kraftigt sätt att integrera hela kedjan av bygg-, test- och distribueringsverktyg (Heller, 2017).

3. KRAV

3.1 Vad ingår i projektet?

Tillsammans med min projektbeställare och handledare tog jag fram en kravspecifikation över projektet. Här listar jag den funktionalitet som ska ingå i mobilapplikationen. Det framkom även att denna funktionalitet kan ändras och byggas på under projektets gång.

Rent funktionalitetsmässigt ingår:

- Val av meny i applikationen
- Inloggning till applikationen
- Val av design/layout av applikationen

Val av meny innebär att användaren ska kunna välja att navigera i applikationen med hjälp av en sidomeny eller med hjälp av flikar. Inloggning innebär att det inte ska fungera att hämta en viss sida utan att först ha loggat in. Crosskey har hittills tre olika autentiseringsmetoder, vilket man vill kunna visa upp i applikationen. Val av design och layout innebär att det finns skräddarsydda teman att välja mellan.

Andra krav som implicit ingår är att applikationen utvecklas med kombinationen Angular och Ionic, att den har en egen Git förvaringsplats och egna Jenkins byggen.

4. EN FÖRSTA GRUND

4.1 Bakgrund

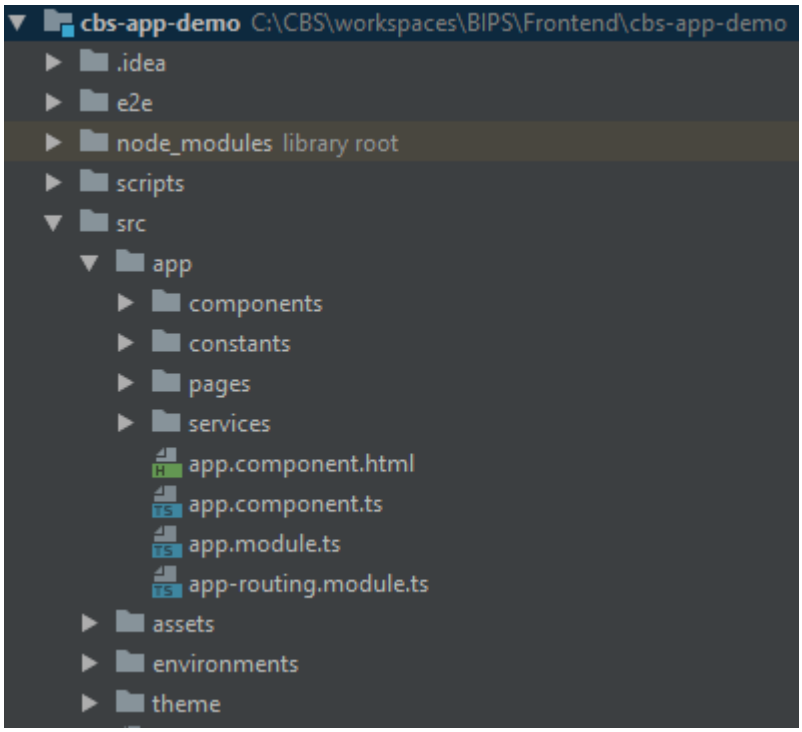
När ett utvecklingsprojekt planeras så väljs programmeringsspråk och utvecklingsmiljö rätt tidigt. I det här fallet så var det från början bestämt att Angular och Ionic skulle användas. Men vad är det som gör att denna kombination är ett bra alternativ för moderna hybridapplikationer? Ett Angular-projekt består av komponenter, vilket ger fördelar som utvecklingsflexibilitet och separation av CSS-hantering (Rangpariya, 2019). Användning av komponenter har gradvis tagits över inom frontend-utveckling.

Det är ännu vanligt att det utvecklas i Java/Kotlin för Android-applikationer och i Obj-C/Swift för iOS-applikationer. Ionic används för att utveckla hybridapplikationer, vilket ger stöd för både native och webb (Rangpariya, 2019). Ionic har många inbyggda komponenter som är lätta att använda, från formulärelement till olika former av popup-dialoger. Ionic tillhandahåller även ett API som introducerar vanliga och viktiga funktioner av en sidas livscykel, exempelvis.

Tillsammans så är Angular och Ionic en stark kombination som underlättar arbetet att utveckla applikationer för både Android och iOS. Att skapa upp ett Angular- och Ionic-projekt är väldigt enkelt. Både Angular och Ionic tillhandahåller enkla tutorialer för att skapa upp en första början till en applikation. Sedan kan applikationens sidor och funktionalitet byggas upp.

4.2 Utveckling av applikationen

Vid skapandet av projektet så skapas en struktur automatiskt enligt Angulars standard. Modifikationer går att göra och i detta fall så har Crosskey en egen standardstruktur på sina projekt som jag väljer att följa (se Figur 2).



Figur 2. Prototyp-applikationens struktur.

I strukturen finns bland annat substrukturer för komponenter, konstanter, sidor, tjänster, tillgångar och tema. Komponenter är återanvändbara element. Det kan exempelvis vara en knapp som är särskilt stylad och ska användas för ett specifikt ändamål. Konstanter kan ses som variabler, som även de används där de behövs i applikationen. Under "pages" hittas HTML-filer med tillhörande logik för applikationens olika sidor.

Strukturen tjänster innehåller generella tjänster som applikationen kan ta del av. Här valde jag att lägga till en understruktur för språk och mockhantering. Tillgångar och tema tillhandahåller resurser som språk, bilder och stylingfiler.

När strukturen är etablerad var det bara att börja utveckla. Ett krav var att applikationen ska vara flexibel och dynamisk. Därför började jag på en startsida där användaren ska exempelvis kunna välja vilka sidor som ska finnas med. Dessa sidor hade jag diskuterat med min beställare om. De kan vara exempelsidor som "Betala" eller "Konton", vilket passar en bankapplikation, eller sidor mera med syfte att exempelvis visa grafiska diagram. När användaren väljer de sidor som ska

finnas, sparas de i en tjänst som jag sedan kan använda för att hämta dessa sidor och visa dem i en meny.

Som tidigare nämnt så byggs ett Angular-projekt upp med hjälp av användning av komponenter. Det kan låta ganska okomplicerat, men den stora poängen är att så långt som möjligt skapa återanvändbara komponenter. I allmänhet består en applikation av ett antal "dumma" komponenter och ett antal "interaktiva" komponenter. De interaktiva komponenterna reagerar på en användares val och ändrar sin egna status, medans dumma komponenter inte gör det. De komponenter som är mest återanvändbara är de "dumma" komponenterna, eftersom de fokuserar på en enda sak och gör inga beslut (di Lauro, 2018).

Detta betyder dock inte att alla komponenter måste vara "dumma", det är en övervägning som måste tas av utvecklarna. Men det är bra att ha i åtanke att en komponent antagligen inte ska vara särskilt stor och ha för mycket ansvar. Figur 3 visar ett exempel på en Angularkomponent.

src/app/hero-list.component.html

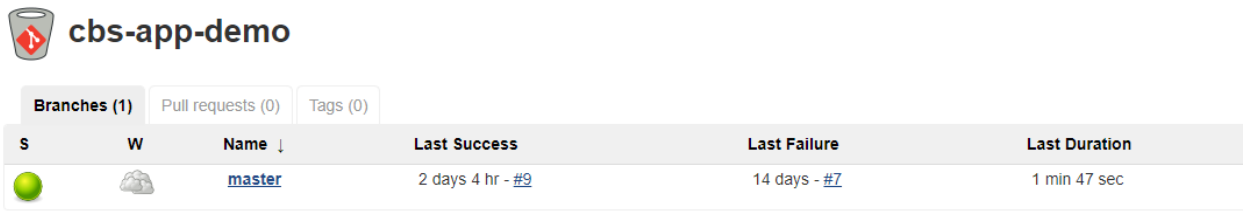
```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

Figur 3. Exempel på en Angularkomponent (Angular.io 2020).

4.3 Skapande av Jenkinsbygge

När jag fått en god start på utvecklingen av applikationen, så kunde jag skapa ett Jenkinsbygge för den. Med hjälp av min handledare så skapade vi upp de filer som behövdes. I en så kallad Jenkinsfile kunde vi definiera vilka steg som bygget behöver utföra. De steg som behövs i nuläget var npm install- och build-stegen. Npm install är ett Node.js kommando som installerar de beroenden som projektet har till olika bibliotek och verktyg. Build är sedan huvudsteget som kompilerar projektet.

Några andra scriptfiler till behövdes, men det tog inte länge innan vi hade fixat ett fungerande Jenkinsbygge. Figur 4 visar mastergrenen av demoapplikationens projekt.



Figur 4. Jenkinsbygget för demoapplikationen.

5. HANTERING AV MOCKDATA

En fråga som jag fick tackla ganska snabbt var hur mockdata skulle lagras och användas i applikationen. Mockdata behövs eftersom mobilapplikationen inte kommer arbeta med/mot någon backend.

5.1 Val av metod

Det finns flera olika möjligheter för att få detta att fungera (Kuehnel, 2016), exempelvis:

- Statiska JSON-filer via Ajax
- Online mocking-tjänster
- JSON-server
- Egen server

Det första tillvägagångssättet med statiska JSON-filer är det enklaste alternativet. De falska data skapas och lagras direkt i filsystemet av projektet. Fördelarna med detta sätt är att det är okomplicerat och att data finns väldigt lätt tillgängligt. Nackdelarna däremot är att det är mycket begränsat vad gäller funktionalitet, den enda HTTP-metod som fungerar är GET.

Online mocking-tjänster innebär dedikerade tjänster som erbjuder lösningar som hjälper dig att mocka dina HTTP-svar. Detta alternativ är enkelt att använda och ger en större möjlighet att konfigurera lösningen. Datat lagras dock på publika servrar som du inte äger och ditt projekt blir beroende av leverantören av tjänsten.

Alternativet att ha en JSON-server innebär att de data man behöver serveras från filsystemet via en lokal server. Fördelarna är att det är enkelt att anordna, det stödjer de mest använda HTTP-verben och dina ändringar sparas till din JSON-källa med POST-, PUT-, PATCH- och DELETE-förfrågningar. Men även denna lösning har sina nackdelar: det går inte att ha självdefinierade dynamiska delar av en API-väg (exempelvis api/articles/{id}) samt underhåll av en enda JSON-fil för alla dina svar.

Det fjärde sättet att lösa "mocking" frågan på är med en egen server. Detta val är det mest komplexa och är ett bra alternativ om det är möjligt att ha en rimlig kompromiss mellan flexibilitet och komplexitet. Ett exempel till lösning som går att hämta från GitHub är "HTTP-fake-backend", som erbjuder en server att bygga upp ett falskt applikationsprogrammerings-gränssnitt (Kuehnel, 2016).

Det sistnämnda alternativet var det som jag bestämde mig för att testa först. Men ett problem med "HTTP-fake-backend" var att servern behövde byggas upp skiljt från applikationen. Eftersom det inte är tänkt att det ska vara komplicerat att starta applikationen och köra den var som helst, blev det snabbt bestämt att detta inte var lösningen som jag letade efter.

Mitt nästa val var att prova JSON-server alternativet. Efter en del undersökning hittades "Angular in-memory-web-api", som är ett API som fångar upp HTTP-förfrågningar och dirigerar om dem till ett eget datalager (NPM, 2019). Användningsfallen består bland annat av dessa:

- Demoapplikationer som behöver simulera operationer som ger persistens av data utan en riktig server. Ingen testserver behövs.
- Skapande av prototyper och koncepttest (PoC)
- etc.

Detta verkade vara ett bra alternativ för mitt ändamål, så jag valde att installera och prova på API:et.

5.2 Databas i minnet

Det första som behöver göras efter att man har installerat tjänsten är att skapa en klass som implementerar "InMemoryDbService". Denna klass kan ses som själva databasen för den information som ska lagras. Det som sedan måste göras är att implementera funktionen "createDb()". Här samlas all mockdata, som är räckor av objekt som ska returneras eller uppdateras. Nedan följer ett exempel (se Figur 5).

```
import { InMemoryDbService } from 'angular-in-memory-web-api';

export class InMemHeroService implements InMemoryDbService {
  createDb() {
    let heroes = [
      { id: 1, name: 'Windstorm' },
      { id: 2, name: 'Bombasto' },
      { id: 3, name: 'Magneta' },
      { id: 4, name: 'Tornado' }
    ];
    return {heroes};
  }
}
```

Figur 5. Exempel på implementation av "InMemoryDbService" (NPM, 2019).

För att få tag på den lagrade informationen behövs tjänster som med hjälp av HTTP-anrop hämtar det som efterfrågas. En tjänst kan då skapas, som innehåller dessa anrop som att hämta alla hjältar eller en specifik hjälte med hjälp av id. Det fungerar även att skapa ett hjälteobjekt och lägga till det i databasen, detsamma med att ta bort ett hjälteobjekt.

Med hjälp av denna tjänst så kunde jag skapa mockdata av kunder, bankkonton och betalningar. Jag skapade modeller av enskilda objekt, exempelvis en kund, och en tjänst för att hämta ut kunder ur databasen. Figur 6 visar en del av CustomerService, tjänsten som hämtar kunddata.

```
@Injectable()
export class CustomerService {
    endpoint = 'customers';
    customer = new Subject<CustomerModel>();

    constructor(private http: HttpClient) {
    }

    getCustomers() {
        return this.http.get( url: API.base_url + this.endpoint);
    }

    getCustomer(id) {
        return this.http.get( url: `${API.base_url + this.endpoint}/${id}`)
            .pipe(map( project: customer => {
                this.customer.next(<CustomerModel>customer);
                return customer;
            }))
            .pipe(catchError( selector: error => {
                return of (error);
            }));
    }
```

Figur 6. En del av CustomerService.

Anropen till dessa tjänster går mycket snabbare än ett riktigt API-anrop till backend, vilket är en stor fördel för denna demoapplikation.

6. DESIGN OCH LAYOUT

En rätt viktig del i byggandet av mobilapplikationen är designandet av den. Det bör nämnas att designen inte behövde vara fullständig för detta projekt. Ett önskemål från beställaren var att det åtminstone ska kunna gå att ändra tema i applikationen. Detta kan uppnås med hjälp av Ionic ramverket och CSS-variabler.

6.1 Teman

Genom att skapa CSS-variabler i pseudoklassen :root, så kommer de att finnas tillgängliga i hela applikationen för alla element (Stelzer, 2018). Detta kan vi utnyttja genom att skapa temaklasser i :root, se Figur 7.

```
:root {
  .red-theme {
    // your colors --ion-color-primary etc
  }
}
```

Figur 7. Exempel på ett rött tema i :root (Stelzer, 2018).

Nästa steg är att skapa en tjänst som kan lägga till ett valt tema till bodytaggen. Där ska en metod för att lägga till ett tema och en metod för att ta bort ett tema finnas. Sedan är det bara att anropa dessa två funktioner när användaren exempelvis klickar på en knapp. För demoapplikationen så skapade jag en sådan knapp direkt på startsidan, se Figur 10.

6.2 Navigering

En annan designfråga som diskuterades med beställaren var hur användaren ska navigera sig i applikationen. Ett vanligt alternativ är med hjälp av en sidomeny, men beställaren ville inte låsa sig till endast detta sätt. Användaren ska kunna välja mellan sidomeny eller flikar.

För att lösa detta behövde jag skapa ännu en till tjänst för att hålla reda på vilket val som användaren hade gjort. Sidomeny skapades med hjälp av Ionics ion-menu-komponent (Figur 8) och flikarna kunde jag skapa med hjälp av ion-tabs-komponenten (Figur 9).

```
<ion-menu side="start" menuId="first">
  <ion-header>
    <ion-toolbar color="primary">
      <ion-title>Start Menu</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    <ion-list>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
      <ion-item>Menu Item</ion-item>
    </ion-list>
  </ion-content>
</ion-menu>
```

Figur 8. Ion-menu-komponenten (Ionic Framework, 2019).

```
<ion-tabs>
  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="schedule">
      <ion-icon name="calendar"></ion-icon>
      <ion-label>Schedule</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
</ion-tabs>
```

Figur 9. Ion-tabs-komponenten (Ionic Framework, 2019).

Även själva navigeringsalternativet kan väljas på startsidan jag skapade. När användaren väljer antingen “sidomeny” eller “flikar”, så sparas valet i Ionic storage. Ionic storage tillhandahåller ett enkelt sätt att lagra nyckel/värdepar och JSON objekt. I detta fall så lagrar jag ett

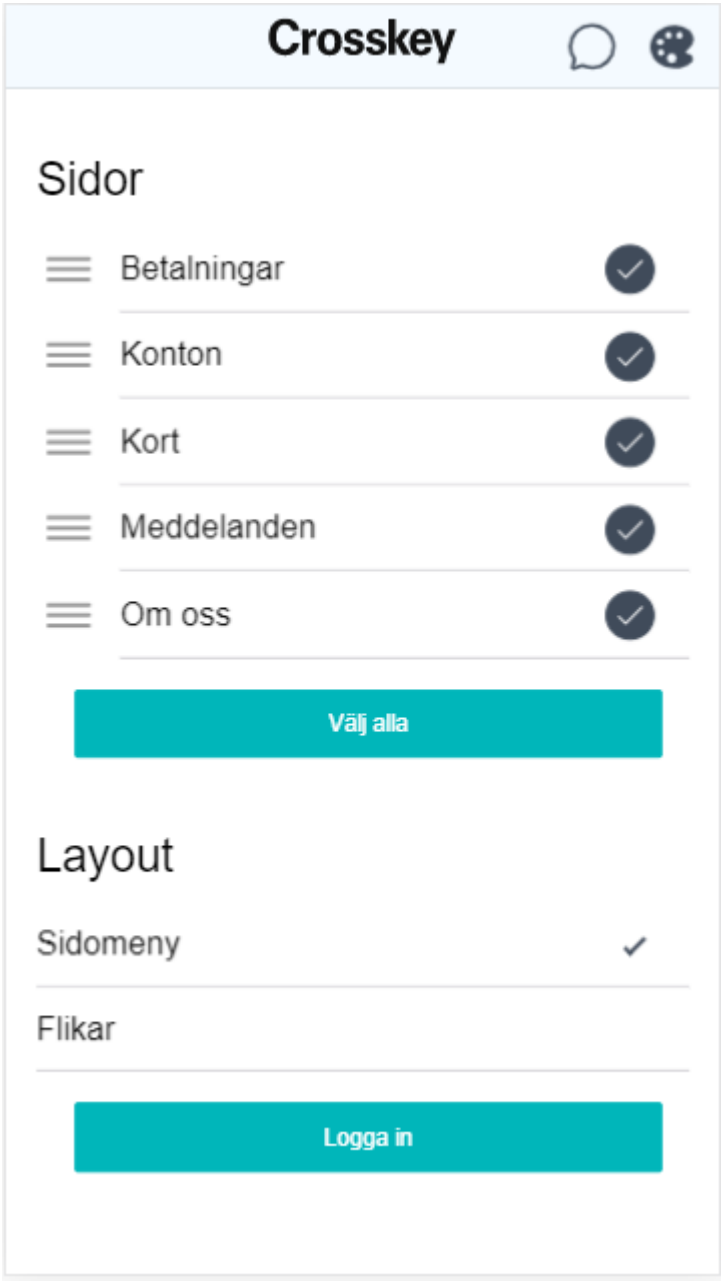
nyckel/värdepar, där värdet antingen är “MENU” eller “TABS”. Sedan kan jag kolla detta värde när användaren loggar in, och på det sättet bygga upp layouten med en sidomeny om det hade valts.

6.3 Val av språk

Jag valde att även implementera ett sätt att välja mellan tillgängliga språk i applikationen. Sättet jag implementerade detta på hade jag stor hjälp från Crosskeys sida, eftersom deras applikationer gör på samma sätt. Med hjälp av NGX-Translatebiblioteket för Angular så kan man definiera översättningar i olika språk och lätt välja mellan dem (NGX-Translate, 2017).

6.4 Resultat

Figur 10 visar den slutliga startsidan för demoapplikationen från detta projekt. Högst upp går det att välja språk via språkbubblan och tema via paletten. De sidor jag skapat under projektet listas upp och användaren kan välja vilka som ska synas efter inloggning. Användaren kan även välja mellan sidomeny eller flikar som navigeringsmöjlighet.



Figur 10. Demoapplikationens startsida.

7. SLUTSATSER

Syftet med arbetet var att ta fram en demomobilapplikation för Crosskey Banking Solutions, som kan demonstrera funktionalitet och design för potentiella kunder. Applikationen skulle ha en dynamisk grund som kan vidareutvecklas och kunna användas utan internetuppkoppling. En första version har tagits fram av en sådan applikation.

7.1 Framtida utveckling

Det finns mycket utvecklingsarbete kvar innan applikationen kan användas i ett riktigt säljscenario. Mer funktionalitet och flera teman exempelvis skulle kunna färdigställa demoapplikationen. Undersökning i att integrera applikationen mot Open Banking API:er har redan påbörjats och skulle vara ett gott försäljningsargument för Crosskey.

7.1.1 Open Banking integrering

Open Banking är ett begrepp från finansiella tjänster, som hänvisar till användningen av öppna API:er inom området. Det möjliggör även flera transparenta alternativ för kontoinnehavare vad gäller öppna data till privat data, samt användningen av öppen källkod för att åstadkomma detta (Open Banking, 2020).

För att en koppling mellan demoapplikationen och open banking ska vara möjlig behövs X.509-certifikat för att kunna identifiera sig mot API:erna. Dessa certifikat är även till för att säkra kommunikationen mellan applikationen och API:erna. En tutorial finns för att koppla upp sig mot Crosskeys lösning, där man skapar upp ett konto och ett team för applikationen. Jag har ännu vid denna tidpunkt inte kommit mycket längre, men förhoppningen är att kunna kommunicera med API:erna inne i applikationen och hämta ut Sandbox-data för olika testkunder.

7.2 Reflektioner

Detta uppdrag har varit ett kul utvecklingsarbete för mig och jag har lärt mig mycket om Angular och Ionic hos Crosskey. Det har varit intressant att utveckla en app från grunden och strukturera upp projektet på ett förståeligt sätt.

Den kunskap om webbutveckling jag hade från innan var främst HTML, CSS och Javascript. Efter att ha arbetat hos Crosskey så har jag fått kunskap om hybridapputveckling med Angular och Ionic. Min förhoppning är att fortsätta lära mig mera inom området och testa på flera intressanta programmeringsspråk och verktyg.

KÄLLOR

Angular (2019, 10 september). I *Wikipedia*. Hämtad från https://en.wikipedia.org/wiki/Angular_(web_framework)

Angular.io (2020). Introduction to components and templates. Hämtad från https://angular.io/guide/architecture-components

Griffith, C. (2019). What is Hybrid App Development? Hämtad från https://ionicframework.com/resources/articles/what-is-hybrid-app-development

Hamilton, R. (2016). Data mocking - A way to test the untestable. Hämtad från https://blog.scottlogic.com/2016/02/08/data-mocking.html

Heller, M. (2017). What is Jenkins? The CI server explained. Hämtad från https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html

Ionic (2019, 24 september). I *Wikipedia*. Hämtad från https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)

Ionic Framework. (2019). API Index. Hämtad från https://ionicframework.com/docs/api

Kuehnel, M. (2016). Data mocking - ways to fake a backend (API). Hämtad från https://michael-kuehnel.de/api/2016/11/04/data-mocking-ways-to-fake-a-backend-api.html

di Lauro, A. (2018). Angular patterns 2: how to write seriously reusable components. Hämtad från https://itnext.io/angular-patterns-2-how-to-write-seriously-reusable-components-96be16568abc

NGX-Translate. (2017). The internationalization (i18n) library for Angular. Hämtad från https://www.ngx-translate.com/

NPM. (2019). Angular in-memory-web-api. Hämtad från https://www.npmjs.com/package/angular-in-memory-web-api

Open Banking (2020, 11 februari). I *Wikipedia*. Hämtad från https://en.wikipedia.org/wiki/Open_banking

Rangpariya, N. (2019). How Does Angular + Ionic Make The Best Hybrid Applications. Hämtad från https://medium.com/swlh/how-does-angular-ionic-make-the-best-hybrid-applications-467038ec8767

Stelzer, P. (2018). Ionic 4: The Ultimate theme switcher. Hämtad från https://medium.com/@paulstelzer/ionic-4-the-ultimate-theme-switcher-ab5a592cb1c4